B8 cont

the first time. The attribute, m_FormatProtocolVectorMapIterator, is used by the getFormatAndProtocolVector function to iterate over the m_FormatProtocolVectorMap. Figures 23C and 23D show the steps for three main functions in the CFormatProtocol_InformationBase class 600.

Please delete paragraph [p115] and substitute therefore:

B9

Figures 25A, 25B, 25C, 25D, and 25E show the class definition of the CProtocolRestrictionCheck class 620. The m_bOneFormatRestriction attribute specifies whether the information is restricted to a single format. Other types of restrictions could be implemented by adding other private functions and attributes. An exemplary restriction algorithm for the present invention is illustrated in Figures 25C, 25D, and 25E (showing the steps in the private function oneFormatRestriction).

REMARKS

Favorable reconsideration of this application in light of the following discussion is respectfully requested. Claims 1-9 and 11-20 are pending in the present application. No claims have been amended herewith.

The specification has been amended to correspond with the formal drawings that were previously filed on May 27, 2003. The preparation of formal drawings required additional sheets of drawings, as the text on the figures was enlarged to comply with the draftsperson's requirements. For example, Figures 23A-23B became Figures 23A-23D, and the specification was amended accordingly.

Consequently, in view of the present amendment and in light of the above discussions, the outstanding grounds for rejection are believed to have been overcome and in condition for allowance. An early and favorable action to that effect is respectfully requested.

Respectfully submitted,

OBLON, SPIVAK, McCLELLAND,
MAIER & NEUSTADT, P.C.

[illegible]

Gregory J. Maier
Registration No. 25,599
Attorney of Record
Michael R. Casey, Ph.D.
Registration No. 40,294

22850
(703) 413-3000
Fax (703) 413-2220

Marked-Up Copy

Serial No. 09/453,937
Amendment Filed: 6-18-03

IN THE SPECIFICATION

Please amend the specification as follows:

Please delete paragraph [p37] and substitute therefore:

Figures 18A, 18B, 18C, 18D, 18E, 18F, 18G, and 18H [through 18D] show a Processor Builder class according to one embodiment of the present invention;

Please delete paragraph [p42] and substitute therefore:

Figures 23A, 23B, 23C, and 23D [and 23B] are class specification of CFormatProtocol_InformationBase class that interface with the System Manager computer code device;

Please delete paragraph [p44] and substitute therefore:

Figures 25A, 25B, 25C, 25D, and 25E [25A, 25B and 25C] are class specification of CProtocolRestrictionCheck class where the steps in the oneFormatRestriction function shows the process to modify the map structure; and

Please delete paragraph [p107] and substitute therefore:

Figure 17 shows the map data structure (using (key, value) pairs) used for creating and caching the protocol processor. In one embodiment, for a given formatted event data, multiple protocols may need to send the same event data. Likewise, the same protocol may be requested to send the same data using different data formats. As discussed above, the key of each pair specifies a protocol (e.g., SMTP, FTP or HTTP) to be used. The value of the map is itself a second pair (x,y), where x is a pointer to the protocol processor object, and where y is a pointer to the function that creates the protocol processor. This map is initialized to contain the keys and values with the x value being assigned a flag value (e.g., zero). When a particular protocol is requested, the x value corresponding to the specified key is checked. If the x value is zero, the function pointed to by y value is executed. That execution creates the protocol processor, and the pointer to the created object is stored in the x value. Then, the newly created or subsequently stored pointer to the object is then returned as a pointer to the abstract class. Step 6 of [Figures 16A and] Figure 16B (corresponding to step 8 of Figure 14) illustrates an exemplary code fragment that calls the createProtocolProcessor function. The createProtocolProcessor function returns a pointer to the abstract protocol processor object (identified by the return type "CAbsProtocolProcessor *"). Step 7 of Figure 16B (corresponding to the step 9 of Figure 14) shows sending monitored information in a requested format.

Please delete paragraph [p108] and substitute therefore:

Figures 18A, 18B, and 18C show [Figure 18A shows] the function list and the attributes of the CProcessorBuilder Class according to one embodiment of the present invention. The public function createDataFormatProcessor receives the specification for the Data Formatter and returns the pointer to the specified Data Formatter object in the abstract class type. The public function createProtocolProcessor function receives the specification for the Protocol Processor and returns the pointer to the specified Protocol Processor in the abstract class type. The m_pDataFormatter attribute of the class is used to cache the specified data formatter in the class. The other two map attributes show the structure shown in the Figures 15 and 17. The function definition section shows the steps used by the various functions declared in the function list.

Please delete paragraph [p110] and substitute therefore:

Figure 20 shows relationships of the CFormatProtocolCombinationCheck class 610 and the CProtocolRestrictionCheck class 620 used within the Format And Protocol Information Base System. Generally, the CFormatProtocol_InformationBase interface 600 (described in more detail with reference to Figures 23A, 23B, 23C, and 23D [and 23B]) keeps track of the specified formats and protocols. That interface 600 uses the (1) CFormatProtocolCombinationCheck class 610 and (2) the CProtocolRestrictionCheck class 620 for (1) verifying requested combinations and (2) checking for restrictions on the protocols, respectively.

Please delete paragraph [p111] and substitute therefore:

Figure 21 describes the process by which the system manager 560 verifies whether a specified format and protocol combination is valid. A storeFormatAndProtocol request from the system manager 560 is initially handled by the CFormatProtocol_InformationBase interface 600. Using the relationships illustrated in Figure 20, that interface 600 converts the request to an isFormatProtocolCombinationOK request that is sent on to the

CFormatProtocolCombinationCheck class 610. If that class 610 returns "true," then the combination is valid; otherwise, the class 610 returns false indicating that the combination is invalid. When the combination is valid, the two values are stored into two different maps specified in the Figures 23A and 23B [Figure 23A].

Please delete paragraph [p113] and substitute therefore:

Figures 23A and 23B are [Figure 23A is] an exemplary class definition of CFormatProtocol_InformationBase interface/class. The functions, storeFormatAndProtocol and getFormatAndProtocolVector, are public functions used by the System Manager 560 computer code device (an object of CMonitorManager class). Two map structures keep the specified formats and protocols passed through the function, storeFormatAndProtocol, after checking the validity of the combination of the format and the protocol through the object, m_FormatProtocolCombinationCheck of CFormatProtocolCombinationCheck class. The class also contains the object, m_ProtocolRestrictionCheck of CProtocolRestrictionCheck class. The flag, m_bFirstGetCall is used to call the function in the m_ProtocolRestrictionCheck when the function getFormatAndProtocolVector is called for the first time. The attribute, m_FormatProtocolVectorMapIterator, is used by the getFormatAndProtocolVector function to iterate over the m_FormatProtocolVectorMap. Figures 23C and 23D show [Figure 23B shows] the steps for three main functions in the CFormatProtocol_InformationBase class 600.

Please delete paragraph [p115] and substitute therefore:

Figures 25A, 25B, 25C, 25D, and 25E [25A, 25B, and 25C] show the class definition of the CProtocolRestrictionCheck class 620. The m_bOneFormatRestriction attribute specifies whether the information is restricted to a single format. Other types of restrictions could be implemented by adding other private functions and attributes. An exemplary restriction algorithm for the present invention is illustrated in Figures [25B and] 25C, 25D, and 25E (showing the steps in the private function oneFormatRestriction).